IBM Developer
SKILLS NETWORK

HTTP Requests in R

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Understand HTTP
- Handle the HTTP Requests and response using R

Table of Contents

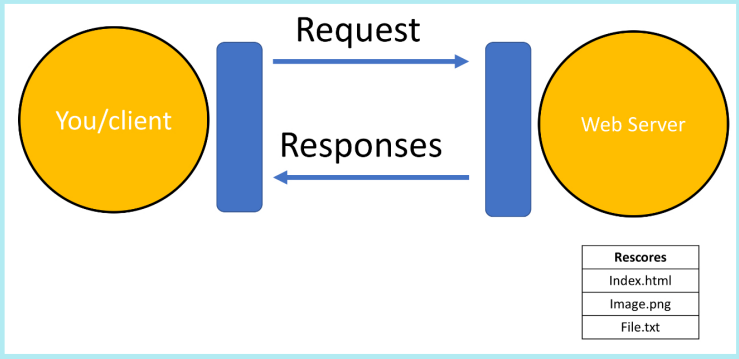
Overview of HTTP

When the **client** uses a web page your browser sends an **HTTP** request to the **server** where the page is hosted. The server tries to find the desired **resource** such as the home page (index.html).

If your request is successful, the server will send the resource to the client in an **HTTP response**; this includes information like the type of the **resource**, the length of the **resource**, and other information.

The figure below represents the process; the circle on the left represents the client, the circle on the right represents the Web server. The table under the Web server represents a list of resources stored in the web server. In this case an `HTML` file, `png` image, and `txt` file .

The HTTP protocol allows you to send and receive information through the web including webpages, images, and other web resources.



Uniform Resource Locator:URL

Uniform resource locator (URL) is the most popular way to find resources on the web. We can break the URL into four parts.

- scheme this is this protocol, for this lab it will always be `http://`
- Internet address or Base URL this will be used to find the location here are some examples: `www.ibm.com` and `www.gitlab.com`
- route location on the web server for example: `/images/IDSNlogo.png`
- URL parameters parameters included in an URL for example: `?userid=1`

You may also here the term uniform resource identifier (URI), URL are actually a subset of URIs. Another popular term is endpoint, this is the URL of an operation provided by a Web server.

Request

The process can be broken into the request and response process.

The request using the get method is partially illustrated below. In the start line we have the `GET` method, this is an `HTTP` method. Also the location of the resource `/index.html` and the `HTTP` version.

The Request header passes additional information with an `HTTP` request:

Request Message

Request Start line	Get/index.html HTTP/1.0
Request Header	User-Agent: python-requests/2.21.0 Accept-Encoding: gzip, deflate :

When an `HTTP` request is made, an `HTTP` method is sent, this tells the server what action to perform.

A list of several `HTTP` methods is shown below.

HTTP METHODS	Description
GET	Retrieves Data from the server
POST	Submits data to server
PUT	Updates data already on server
DELETE	Deletes data from server

Response

The figure below represents the response; the response start line contains the version number `HTTP/1.0`, a status code (200) meaning success, followed by a descriptive phrase (OK).

The response header contains useful meta information.

Finally, we have the response body containing the requested file an `HTML` document. It should be noted that some request have headers.

Response Message

Response Start line	HTTP/1.0 200 OK
Response Header	Server: Apache- Cache:UNCACHEABLE
Response Body	<!DOCTYPE html> <html> <body> My First Heading My first paragraph. </body> </html>

Some status code examples are shown in the table below, the prefix indicates the class; these are shown in yellow, with actual status codes shown in white. Check out the following link for more descriptions.

1XX	Informational
2xx	Success
200	OK
3XX	Redirection
300	Multiple Choices
4XX	Client Error
401	Unauthorized
403	Forbidden
404	Not Found

The httr library

`httr` is a R library that allows you to build and send `HTTP` requests, as well as process `HTTP` requests easily. We can import the package as follows (may take less than minute to import):

[ ]:
```
library(httr)
```

You can make a `GET` request via the method `get` to www.ibm.com:

[ ]:
```
url<-'https://www.ibm.com/'
response<-GET(url)
response
```

We have the response object `response` , this has information about the response, like the status of the request. We can view the status code using the attribute `status`

[ ]:
```
response$status
```

You can also check the headers of the response

```
response_headers <- headers(response)
response_headers
```

We can obtain the date the request was sent using the key `Date`

```
response_headers['date']
```

`Content-Type` indicates the type of data:

```
response_headers['content-type']
```

To obtain the original request, you can view it via response object:

```
response$request$headers
```

Coding Exercise: in the code cell below, find the content-length attribute in the response header

```
# Write your code below. Don't forget to press Shift+Enter to execute the cell
```

▼Click here for the solution

```
response_headers['content-length']
```

Now, let's get the content of HTTP response

```
content(response)
```

which is the IBM home page (in fact, HTML page which you will learn later in this course)

You can load other types of data for non-text requests like images, consider the URL of the following image:

```
image_url<-'https://gitlab.com/ibm/skills-network/courses/placeholder101/-/raw/master/labs/module%201/images/IDSNlogo.png'
```

We can make a get request:

```
image_response<-GET(image_url)
```

We can look at the response header:

```
image_headers <- headers(image_response)
```

We can we can see the `'Content-Type'`, which is an image

```
image_headers['content-type']
```

An image is a response object that contains the image as a bytes-like object. As a result, we must save it using a file object. First, we specify the file path and name

```
image <- content(image_response, "raw")
writeBin(image, "logo.png")
```

Then you should be able to find the `log.png` at the file explorer on the left

---

Coding Exercise: in the code cell below, find another image url and use above code to request and download the image

---

Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. Learn more here.

```
# Find another image URL you are interested, and download the image using above example
```

Get Request with URL Parameters

You can also add URL parameters to HTTP GET request to filter resources. For example, instead of return all users from an API, I only want to get the user with id 1. To do so, I can add a URL parameter like `userid = 1` in my GET request.

Let's see an GET example with URL parameters:

Suppose we have a simple GET API with base URL for `http://httpbin.org/`

```
url_get <- 'http://httpbin.org/get'
```

and we want to add some URL parameters to above GET API. To do so, we simply create a named list with parameter names and values:

```
query_params <- list(name = "Yan", ID = "123")
```

Then passing the list `query_params` to the `query` argument of the `GET()` function.

It basically tells the GET API I only want to get resources with name equals `Yan` and id equals `123`.

OK, let's make the GET request to 'http://httpbin.org/get' with the two arameters

```
response <- GET(url_get, query=query_params)
```

We can print out the updated `URL` and see the attached URL parameters.

```
response$request$url
```

After the base URL http://httpbin.org/get, you can see the URL parameters `name=Yan&ID=123` are seperated by `?`

The attribute `args` of the response had the name and values:

```
content(response)$args
```

Post Requests

Like a `GET` request a `POST` is used to send data to a server in a request body. In order to send the Post Request in Python in the `URL` we change the route to `POST`:

[ ]:
```
url_post <- 'http://httpbin.org/post'
```

This endpoint will expect data as a file or as a form, a from is convenient way to configure an HTTP request to send data to a server.

To make a `POST` request we use the `POST()` function, the list `body` is passed to the parameter `body`:

[ ]:
```
body<- list(course_name='Introduction to R', instructor='Yan')
response<-POST('http://httpbin.org/post', body = body)
response
```

We can see POST request has a body stored in fields attribute

[ ]:
```
response$request$fields
```

There is a lot more you can do check out httr here.

Authors

Hi, this is Yan Luo the author of this notebook.

I hope you found it easy to learn how to HTTP requests in R! Feel free to connect with us if you have any questions.

Other Contributors

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-03-05	1.0	Yan	Initial version created